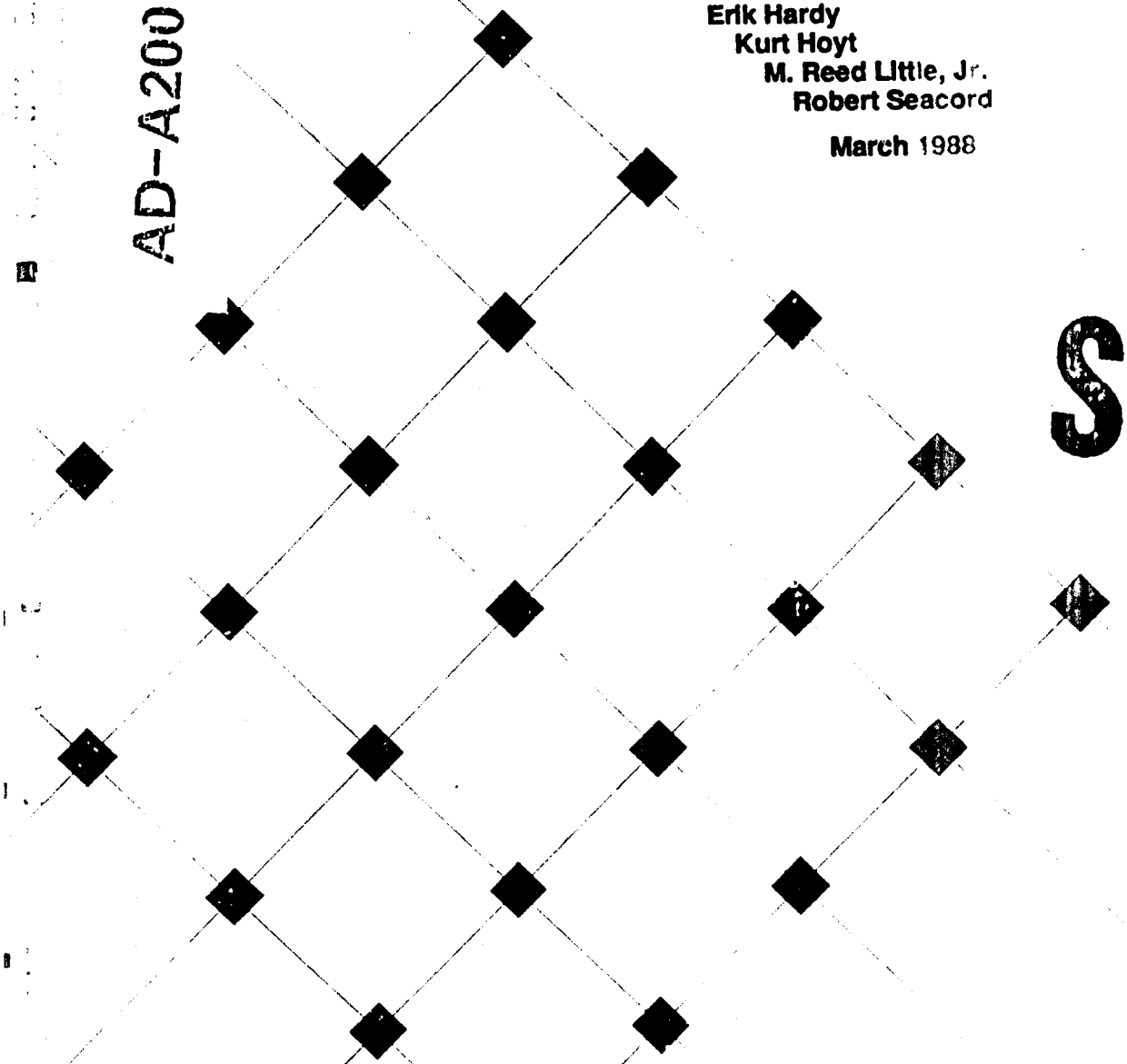
CMU/SEI-88-TR-5
ESD-TR-88-006

Carnegie-Mellon University
Software Engineering Institute

Introduction to the Serpent User Interface Management System

Len Bass
Erik Hardy
Kurt Hoyt
M. Reed Little, Jr.
Robert Seacord

March 1988



SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-88-TR-5	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-TR-88-006
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INST. — 6b. OFFICE SYMBOL (If applicable): SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE-MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE HANSCOM, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE — 8b. OFFICE SYMBOL (If applicable): ESD/XRS1	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962885C0003
8c. ADDRESS (City, State and ZIP Code) CARNEGIE-MELLON UNIVERSITY PITTSBURGH, PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 63752F — PROJECT NO.: N/A — TASK NO.: N/A — WORK UNIT NO.: N/A
11. TITLE (Include Security Classification) Introduction to the Serpent User Interface Management Systems	
12. PERSONAL AUTHOR(S) Len Bass, Erik Hardy, Kurt Hoyt, M. Reed Little, Jr., Robert Seacord	
13a. TYPE OF REPORT FINAL — 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) March 1988 — 15. PAGE COUNT
16. SUPPLEMENTARY NOTATION (Software Engineering Rapid Prototyping Environment)	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

dialogue managers — uims
display design — user interface
prototyping systems — x toolkit

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Serpent is an example of the class of systems known as User Interface Management System (UIMS). It uses the X window system to interact with the end user, and is useful both as a portion of a production system and as a separate prototyping tool. Serpent supports the development and execution of the user interface of a system. It provides an editor with which to specify the user interface and a run-time system which communicates with the application to get the data to display. The system then uses the specification previously output from the editor to decide how to display that data. This report provides a technical overview of Serpent, its components, the module used in specifying the user interface, and the editor used in constructing the user interface.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ OTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED DISTRIBUTION
22a. NAME OF RESPONSIBLE INDIVIDUAL KARL H. SHINGLER	22b. TELEPHONE NUMBER (Include Area Code) 412 268-7630 — 22c. OFFICE SYMBOL SEI JPO

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Technical Report
CMU/SEI-88-TR-5
ESD-TR-88-006
March 1988

Introduction to the Serpent User Interface Management System

Len Bass
Erik Hardy
Kurt Hoyt
M. Reed Little, Jr.
Robert Seacord

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/XRS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

SEI Joint Program Office

Karl H. Shingler
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

Introduction to the Serpent User Interface Management System

Abstract: Serpent is an example of the class of systems known as User Interface Management System (UIMS). It uses the X window system to interact with the end user, and is useful both as a portion of a production system and as a separate prototyping tool. Serpent supports the development and execution of the user interface of a system. It provides an editor with which to specify the user interface and a run-time system which communicates with the application to get the data to display. The system then uses the specification previously output from the editor to decide how to display that data. This report provides a technical overview of Serpent, its components, the module used in specifying the user interface, and the editor used in constructing the user interface.

1. Serpent Overview

Serpent (Software Engineering Rapid Prototyping ENvironmenT) is a system being constructed at the Software Engineering Institute. Serpent will allow the rapid modification of the user interface of an application. It is an example of the class of systems called User Interface Management System (UIMS). These systems are intended to encourage the separation of an application system into a functional portion and a user interface portion, and simultaneously to provide a tool with which to manipulate the user interface portion.

This document introduces concepts behind UIMSs and provides a technical overview of the way in which Serpent treats several of the issues related to UIMSs. It is intended for the technically-knowledgeable reader who is interested in a high-level overview of Serpent. In particular, the following issues are discussed in some detail:

- Interface between Serpent and both the presentation layer and the application
- Model used within Serpent to specify dialogues
- Editor used within Serpent for dialogue specification

The last three sections of this document address the preceding issues. As background for those sections, the two sections that follow discuss briefly the concepts behind UIMSs, the use of Serpent and the flow of control within a system using Serpent.

2. Roles Involved in the Use of Serpent

The purpose of a UIMS, such as Serpent, is to enable the abstraction of information relative to the user interface of an application system, and consequently, to enable the structuring of an application system in a manner which will simplify subsequent modifications to the user interface. The structure of a UIMS becomes evident through an examination of the roles involved in developing and executing the user interface. This structure is examined here in the context of Serpent.

The user interface of an application system is the means by which the system communicates to the end user. This communication is both out of the system to the end user and out of the end user to the system. Serpent is designed around the concept that the *content* of the information communicated to and from the end user is distinct from the *form* of the information. The functional portion of

the application system is concerned with the content of the information and only the user interface portion should be concerned with the form.

This distinction between form and content leads to multiple roles both during the design process and during the execution of the resulting system. During the design process, system designers make fundamental decisions about the content of the information communicated to and from the end user. Decisions must also be made about which tasks belong to the functional portion of the application and which tasks belong to the user interface. Those tasks which generate information (regardless of the form) which is to be communicated to the end user belong in the functional portion, and those tasks which are concerned with the form of the information belong on the user interface side. For example, the functional portion of the application should not be concerned with the language understood by the end user. Dependence on English should be a portion of the user interface.

Once the content of the information is known, then decisions about the structure of the information can be made. These structural decisions about the content of the information available to the end user form the basis of the interface between the functional portion of the application system and the Serpent or user-interface portion. The interface between the functional portion of the application system and Serpent is called *application shared data*, described in Section 3 of this report.

Subsequent to definition of the application shared data the user-interface designer or specifier can begin using Serpent. In concept, these are two separate roles (user-interface designer and user-interface specifier) but one result of using Serpent is that user-interface design can occur during specification.

The user-interface specifier interacts with the Serpent editor to produce the user interface for the application. This is called a *dialogue*. The top portion of Figure 1 shows this interaction.

The designer of the functional portion of the application system, using any methods the designer chooses, completes the design and the implementation of the portion of the application that does not involve the user interface. The total system is now ready for execution. At execution time, the end user sees a complete system with a user interface and the functionality of the application.

The separation of the role of application designer from that of interface designer leads to a system decomposition that has the application as one portion and the user-interface manager as another portion. Serpent plays the role of user-interface manager; Serpent itself can be decomposed also. Serpent has two main portions: The dialogue manager handles the logic of the dialogue and the X window/toolkit system handles both the details of presenting the information to the end user and gathering the end-user interactions with the system. The bottom portion of Figure 1 shows the end user interacting with the X toolkit portion of Serpent, the dialogue controlling the action of the Serpent run-time system (the dialogue manager) and the application. Application, as used in the remainder of this report, means the *portion of the system that does not involve the user interface*.

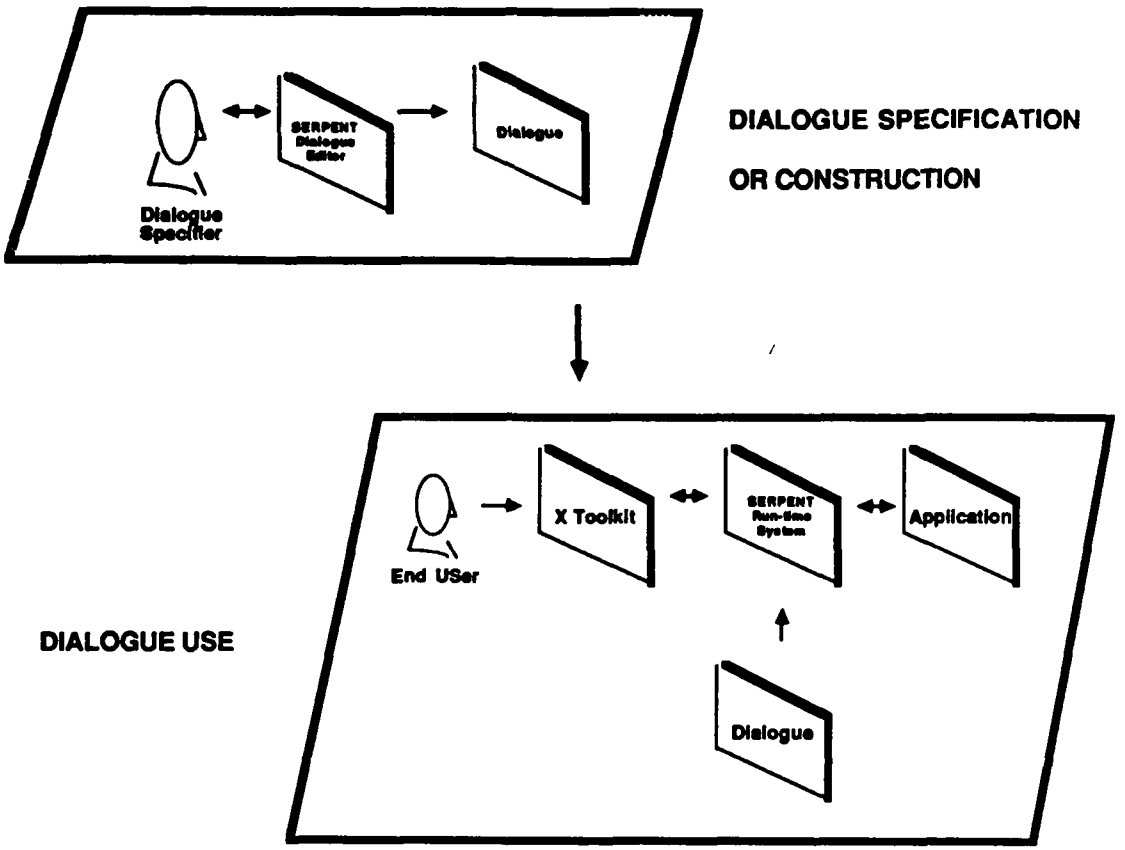


Figure 1: Serpent Dialogue Construction and Use

2.1. Data Flow

Once the application begins executing, data flows through the system as shown in Figure 2. The application places data to be displayed to the end user into Serpent application shared data. The dialogue manager processes the application shared data; the resulting data is shared with the X window/toolkit system. This data is displayed to the end user who interacts with the X window/toolkit until an input is complete and intervention of the dialogue is required. At that time, the X layer places the necessary information into the X shared data and the dialogue manager then processes that information. Depending upon the dialogue, information is either placed back into the X shared data for further presentation, maintained in the dialogue until subsequent actions, or placed into the application shared data. In any case, the appropriate components are notified that data is available for them and the data is then processed.

The preceding data-flow description pertains to data which the end user inputs. The application itself may have other paths external to Serpent for the arrival of data which are not shown in Figure 2.

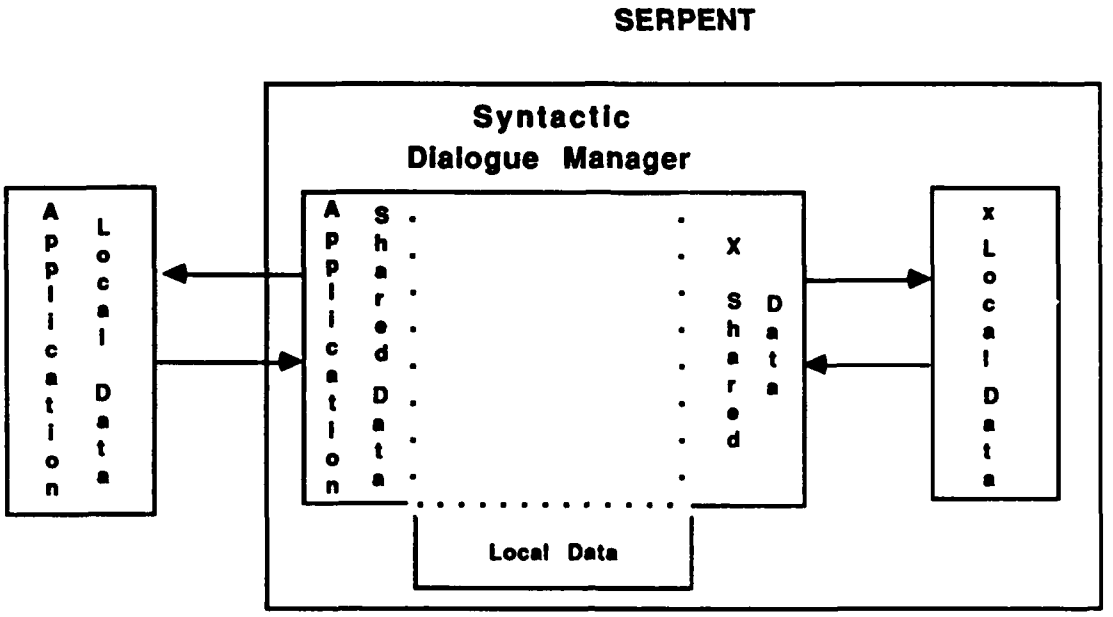


Figure 2: Serpent Data Flow

Notice that two roles of the dialogue are to transform application data into visible objects on the display and to transform user actions into application data. Notice also that by making the data shared between the application and Serpent explicit, it is easier to maintain the separation between the functionality of the application and the user interface.

2.2. Control Flow

An application system using Serpent is conceived of as three independent processes: the application, the dialogue manager and the X window/toolkit. These three processes communicate through the shared data areas as described in the preceding subsections. When the system is linked together, it is possible to specify that these three processes are, in fact, to be executed sequentially; however, Serpent allows for the processes to be independent.

3. Example

The example that follows illustrates the details of Serpent. The display in Figure 3 is adapted from a command and control application. This is the display that the end user of the example sees. The rectangular boxes on the right and left sides (e.g., GS1, GS2) represent sensor sites that detect information regarding site status. The circles in the middle represent correlation centers that collect information from all of the sensors. Each sensor site sends its information to both correlation centers; this explains the duplication of sensor-site boxes on both the right and left sides of the display. The

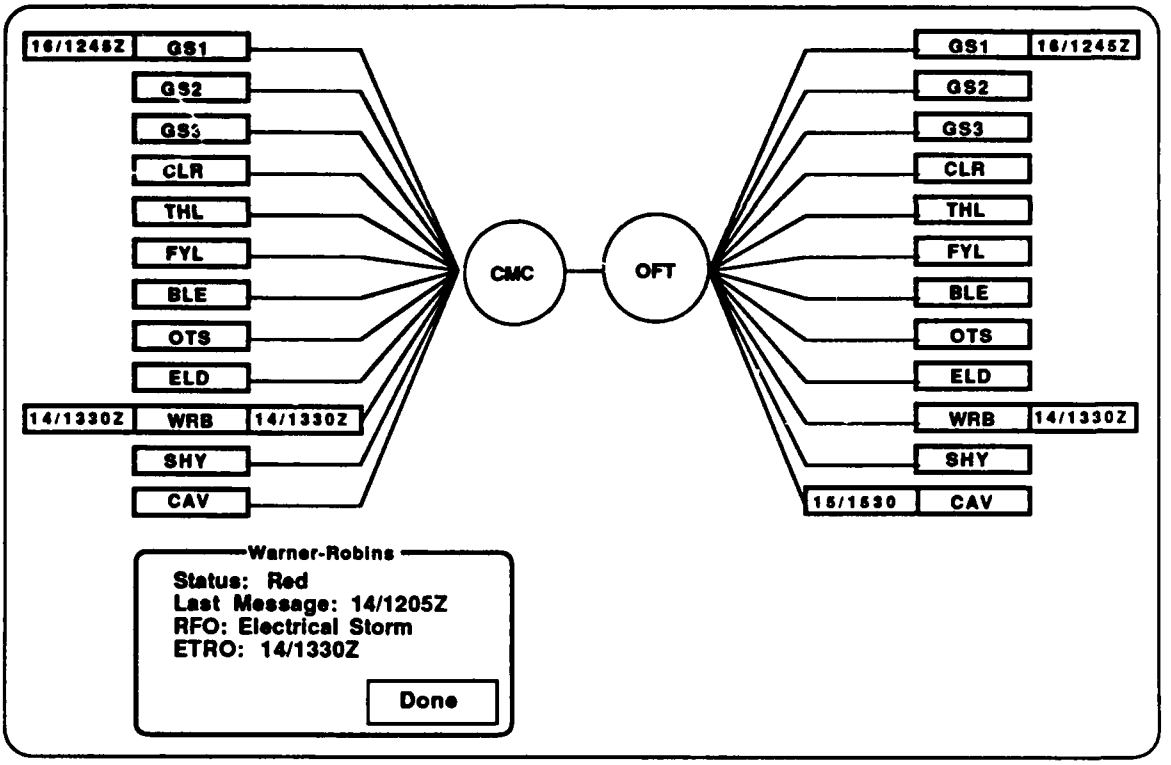


Figure 3: Application Example

lines represent the communication paths between a particular sensor site and a particular correlation center.

When a sensor site is determined to be non-operational, an *estimated time to return to operation*, ETRO, is displayed in association with the site. The ETRO is displayed in association with both occurrences of the sensor site outside of the corresponding boxes. A particular communication line may not be operational, in which case the ETRO for that line is displayed over the line and inside the sensor site box.

The end user may select either one of the sensor site boxes, and a detail window then appears with more status information about the site. This detail window may be edited to modify the ETRO, the status, or the reason for failure. Figure 3 shows the result of selecting the WRB sensor. The RFO, *reason for outage*, is also displayed in the detail window.

Notice that the ETRO for a particular sensor site is always displayed twice. If the user has selected a detail window for the sensor site, the site is displayed three times. This notion that the same piece of information can be displayed multiple times is called *multiple views* of data.

When using Serpent, the application assumes that Serpent is a local database manager that man-

ages the data available to the end user. When this concept is applied to the sensor-site example (see Figure 3), the application functionality relates to converting the information in the local data base of the application into (and from) the information in the data base available to the end user. In Serpent, the data base available to the end user from the application is called *application shared data*. Either the application or Serpent can modify the application shared data. If the application performs the modifications, these are of interest to Serpent; if Serpent performs the modifications, Serpent explicitly informs the application of the changes.

The application that generates the display in Figure 3 maintains a data base of sensor sites, communication lines, and correlation centers. Each component of the data base has associated status information (e.g., ETRO). The application communicates this information to Serpent by writing the information to the application shared data.

Information comes into the application from the user either through Serpent or by direct communications from another computer. A high level outline of the application program is:

- Initialize connection with Serpent.
- Retrieve data from local data base and put into shared data.
- Notify Serpent that data is available.
- Do until exit.
    - Wait for either input from Serpent or message from another site.
    - If input from Serpent then
        - Get updated information from shared data.
        - Verify information.
        - Place updated information into local data base.

    If message from another site then
        - Place new information into local data base.
        - Place new information into shared data.
        - Notify Serpent that new information is available.

Notice in the preceding program outline that the communication with Serpent consists of putting application information (about sensor sites and communication lines) into shared data and then notifying Serpent of new information availability. The application is not aware of the fact that this information is being displayed as a collection of boxes and lines. The application, in the program example, is not aware of the user asking for more detailed information about a particular sensor. This is the essence of the separation of user-interface details from the application program. Notice also that the application is not informed of end user-actions until the actions have changed into the form acceptable to shared data, in other words, until the data is ready for application action.

3.1. Serpent Functionality

The role of Serpent in the sensor-site example (see Figure 3) is to:

- Convert the shared data into the representation of rectangles, lines and circles.
- Determine when the end user selects a sensor site and then display the detail window for that site.

- Convert the modified data into shared data and notify the application of new information, whenever the end user modifies the data being presented.

The dialogue specifier uses the Serpent Editor to construct a dialogue specifying Serpent's role. The dialogue is bound to the application during Serpent initialization. During run time, Serpent uses the dialogue to make decisions about the initial presentation, about modifications to that presentation, and about when to inform the application of the end user's actions.

4. Serpent Shared Data

From an application perspective, data is sent to and from a data base available to the end user and maintained by Serpent. This data is referred to as data shared between Serpent and the application. In fact, application shared data is not made directly available to the end user but is transformed by the dialogue. As discussed previously in this document, when Serpent is decomposed into the dialogue manager and the X window/toolkit, there is another shared-data area for data shared between the dialogue manager and X. Application shared data is transformed into X shared data for interaction with the end user. X shared data is managed in exactly the same fashion as application shared data.

Three elements are involved in shared-data considerations:

1. Data structure
2. Description mechanisms
3. *Timing considerations*

4.1. Data Structure

The example application in section 2 of this document is described as placing data into shared data. Shared data is conceived as a data base of information available to the end user. Shared data can be described as a collection of tables of data. In terms more familiar to programmers, shared data is a collection of records with limited data types used for the components. In the example, one possible structure for some of the data is:

- Sensor site data table
  - site abbreviation
  - site status
  - site full name
  - last message
  - rfo
  - etro

  Communication line data table
  - from sensor site
  - to correlation center
  - status
  - etro

Each table in shared data can be viewed as a collection of rows (*tuples*) or as a collection of columns (*fields*). When either the application or Serpent creates a new row, it is given a unique identification (*shared data element id*). Each field within a table is typed; valid types are: 32-bit integer, 64-bit real, fixed length string, unstructured buffer or shared data element id. Having shared-data element ids within a table allows the dialogue specifier or application designer to build complex data structures using tables within shared data.

4.2. Description Mechanism

A file external to the dialogue and the application contains the description of the shared data's structure. The description is in a special-purpose data-declaration language, *Shared Data Description Language* (SADDLE), tailored especially for Serpent. This file is preprocessed to produce a language-specific description of shared data. Preprocessors exist for Ada and C. Thus, if the application is written in C, the preprocessor will generate *structure* definitions which can be included into the application program. If the application is written in Ada, the preprocessor will generate *package* specifications.

4.3. Timing Considerations

Consider the sensor-site example in section 2 (see Figure 3) again. Data describing the status of a sensor site can originate either from the end user or from another channel of communication and go into the application. Since shared data has both a writer and a reader, and these two are potentially independent processes, it is possible for data to be placed into shared data without the reader necessarily accessing this data immediately. This leads to the following potential timing problem:

1. End user places sensor-site status into application shared data (through actions of the dialogue)
2. Application places different status for same sensor site into application shared data
3. Application notifies Serpent of information to be processed

With the preceding application sequence, the application overwrites the information that the end user has entered. More importantly, the application does not have the possibility of retrieving the data entered by the end user.

To avoid this problem, Serpent uses a transaction model for placing data into shared data. (This solution was chosen over that of requiring all changes to be made simultaneously.) A transaction begins when data is to be placed into shared data: Data then is placed into the transaction and no reader has access to the new transaction until the transaction is committed.. The transaction can be either committed (make data available all at one time) or rolled back (abort the modifications and make no changes to the shared data). Multiple transactions can be active simultaneously.

Transactions have a producer (writer) and a consumer (reader). The writer places requests to create, modify or delete items in shared data and the reader is unaware of any of those requests until the transaction is committed. When a transaction is committed the reader is notified of the modifications to shared data. Since the dialogue manager manages shared data, if the dialogue manager is reading then it can retrieve the data from the transaction and update the shared data base in an atomic fashion. If the reader is the application, then it is being informed of data which has already been applied to application shared data. By informing the reader of the actions in the transaction, it then becomes the reader's responsibility to maintain the information in its own local data area.

Collecting updates to shared data into transactions and applying all of the updates simultaneously prevents problems with losing updates to shared data. By providing access to the updates, both Serpent and the application can act upon every update, even if two transactions modify the same data items.

5. Serpent Model for Dialogue Specification

A Serpent dialogue controls all of the user interface of an application. A dialogue specifier constructs a dialogue using the Serpent Editor. A Serpent dialogue is totally driven by data placed into the various shared-data areas, and is an example of the type of model called *production*. This section discusses the model that the dialogue specifier uses to construct the dialogue; this section also discusses how Serpent interprets the dialogue.

5.1. View Controllers

In Serpent, the actual dialogue between the end user and the application is executed in terms of *view controllers*. A view controller performs two main functions:

- Mapping specific data in the application shared data into objects on the display with which the end user can interact
- Controlling, at a high level, the interactions that the end user has with those objects

A dialogue is specified in terms of view controller *templates*. A template maintains a watch on application shared data for specific data conditions. When data that satisfies a watching view controller template is placed into application shared data, a view controller is created.

The actual view controller has the following functions:

- Tying a particular tuple in shared-data space to the view controller.
- Mapping that data into display objects visible to the end user.
- Performing actions when the end user interacts with the display objects.
- Maintaining local information.

In general, a view controller template consists of five components:

1. Creation condition for a new view controller
2. Actions on creation of a new view controller
3. List of display objects, each object consisting of a collection of attributes for presentation and of methods to respond to end user actions
4. Actions on destruction of created view controller
5. Nested view controller templates

The following subsections discuss the preceding components in general and in terms of the sensor-site example.

5.1.1. View Controllers As Used In Example

In the sensor-site example in Figure 3, there are two rectangles associated with every sensor site (one on the right of the screen and one on the left). For each sensor site, there is a specific tuple in the sensor-site data table within application shared data. This tuple has the information for the sensor site and a view controller maps that tuple into the two rectangles.

Even though there is a separate view controller for each sensor site, there is a single view controller template from which the view controllers are created. This view controller template specifies the condition under which a new view controller is to be created. The created view controller then maps the particular tuple representing a sensor site into the rectangles on the display, interprets any selection of one of these rectangles as the signal to create the detail window view controller, and maintains some local information.

A separate view controller also causes the detail window to be displayed for a sensor site when the end user selects the sensor site. The view controller that controls the selected sensor-site rectangle is informed when the sensor site is selected, and causes the creation of the detail-window view controller.

The view-controller template for the sensor site is:

- Creation condition: sensor site abbreviation is not currently displayed
- Objects:
  - Left sensor-site button (create command button on left side of display)
    - attributes:
      - color
      - size
      - location
      - text in rectangle
    - method
      - select: create view controller that brings up detail box

  Right sensor site button (create command button on right side of display)
    - attributes:
      - color
      - size
      - location
      - text in rectangle
    - method
      - select: create view controller that brings up detail box.

5.1.2. Creation Condition

A view controller template waits until a specific condition is satisfied. In the sensor-site example, the condition is the existence of a new sensor site in application shared data. Once that condition is satisfied, a view controller is created and performs its actions. The creation condition satisfies two purposes. First, it determines when a new view controller is created from a view controller template and second, it associates a tuple from the shared data with the newly created view controller. In the example, when a new sensor site abbreviation is placed into shared data, a view controller is constructed from the template and is associated with the tuple which contains the new sensor site abbreviation. If the view controller exists and the tuple with its particular sensor site abbreviation is deleted, then the view controller ceases to exist.

In general, a view-controller creation condition may be any condition on the attributes in a single shared data table modified by any local information maintained within the dialogue. When the condition is satisfied by certain fields of a particular tuple in a table, then the entire tuple is bound to the view controller. Values other than those participating in the creation condition are typically used for construction of the attributes of objects.

5.1.3. Actions On Creation

Actions on creation are executed when a view controller is created. These actions can be any of the valid ways in which the dialogue can manipulate shared data, manipulate local information, or send information to the application.

No actions on creation are shown in the sensor-site example, but possible actions might be incrementing a count of sensor sites or initializing the flag used to control the display of the detail window.

5.1.4. Objects

Each view controller template describes a collection of objects that are created when a view controller is created from the template. These objects correspond to the shapes on the display and are bound to the newly-created view controller. The objects have attributes that control how they are presented to the end user, as well as methods that determine the high-level interactions that the end user can have with the object. All of the objects within a particular view controller are created when the view controller is created and, thus, the view controller acts as a mechanism for grouping display objects on the basis of the dialogue's logic.

In the sensor-site example, each view controller creates two objects, the right and left sensor rectangles. The attributes of these objects determine the size, location, color and internal text of the objects. The values of the attributes may depend upon the values of the fields in the sensor-site data table tuple that caused the view controller's creation.

In general, the values that the attributes take may depend upon values of the fields in a shared data table or from local information maintained by the dialogue. Several different values can enter into the calculation of a single attribute. In particular, the values can be drawn from the tuple with which the view controller is associated, from values local to the dialogue, or from attributes of other objects. The ability to reference attributes of other objects allows the line objects in Figure 3, for example, to be specified such that they are connected to the sensor-site objects.

The objects also have methods which determine their reaction to end user actions. In the example in Figure 3, the only action an end user can perform on the sensor-site rectangles is to select one of them. The mechanism for the selection is managed by the X toolkit. When the end-user selects a sensor-site rectangle, the X toolkit notifies the dialogue manager that a selection has occurred *for a particular object*. This object belongs to a particular view controller and, consequently, the particular tuple associated with that view controller is known. In the example in Figure 3, when a selection occurs the sensor-site view controller arranges for a detail-window view controller to be created. It could do this by setting a local flag which the detail window view controller template uses as its creation condition.

5.1.5. Actions At Destruction

The view controller is deleted from the system and its objects removed from the display when the creation condition of a view controller becomes false. It is also possible to specify other actions to be performed upon destruction. None are detailed in the example view controller, but possible actions might be to decrement a counter of sensor sites or to inform the application of other information.

5.1.6. Nesting of View Controllers

It is possible to specify that one view-controller template be nested within another view-controller template. This nesting carries through to the actual view controllers created from the templates. A nested view controller inherits the tuple that caused the creation of its predecessor. In the example, the detail-window view controller is nested within the sensor-site view controller. Thus, when the detail-window view controller is created, it inherits the tuple that caused the sensor site view controller to be created. In other words, the detail-window view controller presents certain information about a sensor site and the nesting insures that the information is associated with the correct sensor site. When the sensor-site view controller is destroyed, then all of its nested view controllers, the detail-window view controller in particular, are also destroyed.

5.2. Threads of Control Within Dialogues

A dialogue is specified through a collection of view-controller templates. Each of the view controller templates has a creation condition. The order in which the view controllers are created depends on the data that the application places into shared data and on the actions of the end user. A sub-dialogue is a collection of view controllers that perform one particular task, for example, create the display in Figure 3. It is possible to have multiple sub-dialogues within a dialogue; there are no a priori timing constraints on the execution order for those sub-dialogues.

Actions of the dialogue are determined by the actions of the application and of the end user, and multiple sub-dialogues can be active simultaneously. For example, Figure 3 may represent only one portion of a total display. The end user may select a sensor site and have the detail window displayed, leave it displayed and proceed with an unrelated task in a different portion of the display. Within Serpent, view-controller creation and destruction, as well as methods use, take place completely in response to end-user and application actions. In particular, several sub-dialogues may be carried on in parallel. This allowance of simultaneity of sub-dialogues represents the power of the production model used in Serpent.

5.3. Multiple Views of Data Within Serpent

The data shared between the application and Serpent has one tuple for each collection of application data. The fact that a particular piece of data may be displayed multiple times on a display is reflected only in the dialogue and not in the shared data. In the example, the ETRO for a particular sensor site may be displayed as many as three times. Since the view controllers manage the mapping from application data to presentation objects, Serpent is aware of which view controllers depend upon which values of shared data. Thus, when a particular piece of shared data is modified, Serpent is able to ensure consistency with all of the presentations of that particular piece of shared data.

5.4. Timing of Dialogue Actions

View-controller templates monitor the application shared data area and local dialogue information. Additionally, the templates create view controllers when their creation conditions have been satisfied. Also, when there are changes in the data upon which the objects depend, the new values of the attributes of the objects are recomputed and the object is redisplayed. The arrival of a transaction triggers this activity. Since a transaction may contain modifications to anything in shared data, it is possible for multiple view controllers to be created and attributes of objects to change, all in response to a single transaction. Within the responses to a single transaction, it is also possible for the actions of view controllers to modify data which affect other view controllers. From the perspective of the dialogue specifier, actions in response to a transaction are all simultaneous and, yet, the dialogue manager performs these actions sequentially. Consequently, the timing of the actions of view controllers becomes important.

As long as there is no interaction among the view controllers affected by a transaction, Serpent's actions are equivalent to sequencing through the view-controller templates and active view controllers, and performing all of the actions indicated. The model used, however, has a combined data space consisting of shared data and local data. The creation conditions, the actions and the attribute calculations all treat this data space identically. This allows important functionality, such as, in the example in Figure 3, counting the number of sensor sites and varying the display based on the number of sites exposed to the end user. However, such a treatment of the combined data space creates problems of potential interaction between view controllers. Serpent performs its actions in a fixed order: Their behavior is repeatable, but the correctness of the dialogue is the responsibility of the dialogue specifier.

5.5. Internal Representation

The dialogue is specified in terms of view controller templates. The templates maintain an asymmetrical view of the application shared data. That is, view controllers map application shared data into objects which are to be represented on the display. At a deeper level, however, there really is no asymmetry. The actions of X window/toolkit are translated into X window/toolkit shared data and the dialogue manager is informed of these actions through the same transaction mechanism that the application uses to modify application shared data.[1]

[1]Since this level of Serpent is not visible to the dialogue specifier, it is only mentioned briefly in this report. The interested reader is referred to the system documentation for a thorough discussion of the decomposition and the production system used in Serpent.

For this reason, the editor decomposes view controllers into *productions*. Each production has a condition that causes it to fire; each production also has actions that are executed when the production is fired. The productions operate from a data space and modify the same data space. Therefore, the timing considerations of view controllers also apply to productions.

6. Dialogue Specification

The specification of dialogue is an important aspect of Serpent, because it controls the level of sophistication that the dialogue specifier requires. This section gives an overview of the specification process.

A dialogue is specified through interactions with the Serpent editor. A dialogue has two portions: (1) the displays that the end user sees and uses for interaction, and (2) the logic that determines which display is visible at any instant and controls the interactions with the application. The editor has graphic facilities to specify the components of the display and textual facilities to specify the logic. An important aspect of the process of specifying a dialogue is that whichever is currently the most natural portion of the specification can be worked on at any point in the process. The logic of the dialogue can be specified without regard for the particular displays to be visible and connected with the displays at a later time. Likewise, the visible portion of the dialogue can be specified without regard for the logic and then connected at a later time. Both portions must be specified for a dialogue to execute, but the Serpent editor does not prejudge the order of specification.

The conceptual model that the dialogue specifier must have is that a display is composed of objects, and that the types of objects that compose a display must be of the object types that Serpent supports. Each object has a collection of attributes which determine its appearance. The existing view controllers consist of a collection of objects, and a display is a collection of view controllers. At specification time, there are only view-controller templates, not view controllers, and the person who specifies the dialogue must be aware of that distinction.

The conceptual model leads into how to interact with the editor. One portion of the interaction deals with the visualization of a display, another portion deals with the objects and their attributes, and a third portion deals with the view-controller templates. Each portion has its own abilities to control how the other portions behave. For example, the position attribute of an object can be modified either through explicit modification of the attribute or through movement of the object on the display visualization. The portions of the editor are described in more detail in the subsections that follow.

6.1. Display Visualization

This portion of the editor allows the dialogue specifier to see what a display would look like during execution. It gives a visual presentation of a collection of objects and allows the modification of the geometric aspects of these objects (size and position).

6.1.1. Construction

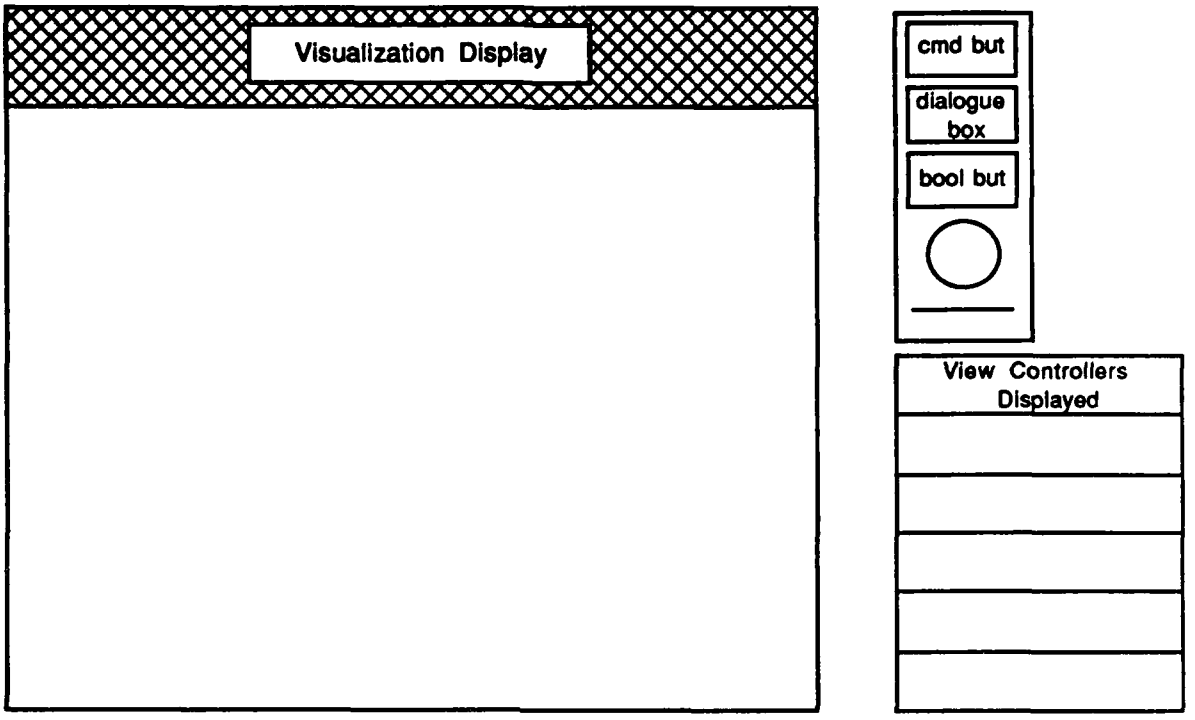


Figure 4: Initial Display

The dialogue specifier is provided a palette of objects with which to construct a planned display. Figure 4 displays the initial portion of the display. The blank window is the target for the construction of the display to be visualized. The specifier chooses elements of the palette and specifies where on the planned display they are to be placed. The specifier also specifies the size of the resulting element. The specification is done using cursor position. Once positioned, the specifier can move or resize objects. Figure 5 displays the desired display after the specifier has entered a command button and a portion of the line connecting the command button with another object.

A number of commands are available to assist in the construction of the planned display. These commands assist in specifying the geometry of the display and in creating objects for the display. These include commands such as:

- Align. Require the x (or y) position of two objects to be the same.
- Copy. Make second distinct copy of object.
- Connect. Require two objects to be connected.
- Group. Specify that two (or more) objects are displayed and positioned relative to each other.

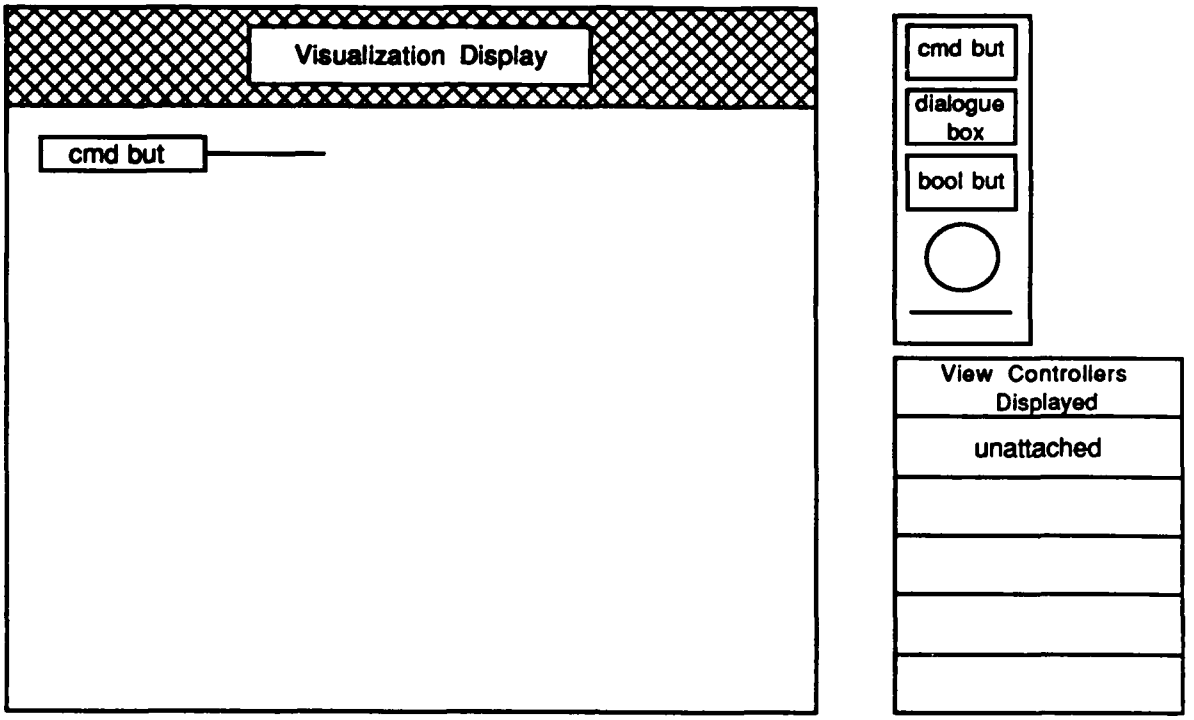


Figure 5: Display After Some Modification

6.1.2. Connection With View Controllers

For completely specified dialogues, objects are associated with view-controller templates. This association can be made either by creating an object as belonging to a particular view controller template, or by associating the object to a view controller template after the object has been created. When an object is created it is associated with a special view controller template given the name *unattached*. An object may be moved from one view controller template to another, and thus moved from the unattached view controller template to the desired view-controller template.

6.1.3. Viewing Display

The target window onto which the objects are moved is composed of all of the objects from a list of view-controller templates. View controller templates can be added and deleted from this list and their objects will appear or disappear accordingly from the visualization display. This allows the dialogue specifier to mimic manually the logic of the display and visually inspect the results of having specific view controllers exist at some point in the dialogue. It also allows the partitioning of the objects into view controller templates. Thus, viewing a particular display contributes to the understanding of how the conceptual structure of a display maps into the view controller templates and into the associated objects.

6.1.4. Connection With Attributes

Each visible object can have its attributes modified directly (see Object Attributes section below). Some attributes can also be modified through the visualization display. These attributes are primarily geometric. It is possible to manipulate directly the size and location of an object, as well as the relation between objects.

6.1.5. Choice of Values

When an object is displayed in the visualization display, it must have values for all of its attributes. Some of these attributes are specified as constants that are easy to assign to the object. Others, however, are specified as depending upon values from shared data, and these values are not available while the dialogue is being specified. Associated with the actual calculation of attributes of objects is a display value which must be a constant and which determines how that attribute is to be assigned during editing. If the display attribute is not given a value, the Serpent editor chooses a value for the purposes of display.

6.1.6. Size of Visualization Display

Objects displayed in the visualization display are shown at actual scale. Typically, there is not enough space on the terminal screen to display the whole scene. The dialogue specifier can set (or change) the size of the window within which the visualization scene is displayed. The specifier can then scroll up and down, as well as left and right, to display any portion of the scene.

6.2. Object Attributes

Each object has a collection of attributes that the dialogue specifier can set. The setting can be constant or vary for the whole dialogue, depending upon the state of the dialogue and the values of shared data. If the setting varies, then it is defined by an expression in the dialogue editor language.

When an object is created it is assigned a system-constructed name. The dialogue specifier can modify that name at any time and the object is referenced by means of that name. Generally, the name given should be related to the function of the object. Nonetheless, by having an initial system-defined name, the actual assignment of the name can be deferred until the workings of the dialogue have been more carefully defined.

An object can have its attributes available for editing through selection from the visualization display, through specification of the edit object from the view-controller template, or through specification of the edit object by name function.

Every object has available a textual-based editor with which to specify the attributes. Figure 6 is an example of that type of editor. Some object types have a specialized graphical editor that uses dials, menus, and so forth, to do the specification. The choice of how to modify an attribute is strictly a matter of convenience; the form of the modification does not affect the result.

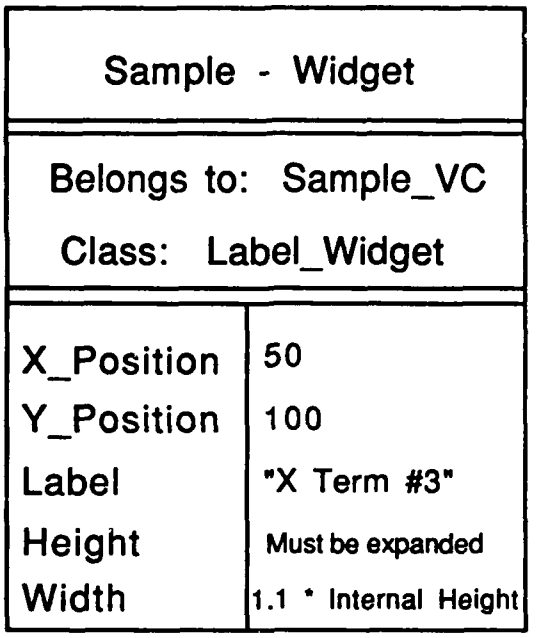


Figure 6: Object Editor

6.3. View-controller Template Specification

Another specialized type of editing function available within the Serpent editor is the editing of view-controller templates. This function can be invoked from the objects on the visualization display (each object belongs to a view-controller template), from a nested or parent view-controller template being edited, or by name from the edit view controller function. The system gives an external name to each view-controller template when the template is created. The name can be modified by the person specifying the dialogue at any point in the dialogue construction and the view controller template can always be referred to by its current name.

View-controller templates have five components: creation conditions, creation actions, possibly nested view controller templates, objects and destruction actions.

The main mechanism for editing view controllers is a textual-based form shown in Figure 6. Each component is entered textually, with the possible exception of objects and nested view controllers. Mouse actions and nested view controllers can associate objects with view-controller templates.
Some view-controller templates refer to a particular shared-data definition. Within the Serpent editor it is possible to view the shared-data definition upon which the dialogue is based. It is not possible, through the Serpent editor, to modify the shared data definition.

7. Summary

Serpent is a system which allows an application to be decomposed into a user interface portion and a functional portion. The separation between the form of the presentation and the function of an application is an important and powerful idea which simplifies subsequent modifications to the presentation. To accomplish this separation, it is necessary to define both a model to describe the user interface and a software architecture which has a separable user interface. This document described the model Serpent uses to describe the user interface and presented the software architecture needed to use Serpent in an application. It also gave an overview of how to specify a dialogue within Serpent.